Clarke and Wright Algorithm

Laboratorio di Simulazione e Ottimizzazione L

M.Battarra, R.Baldacci, D. Vigo

Dipartimento di Elettronica, Informatica e Sistemistica
e
II Facoltà di Ingegneria
Università di Bologna

rev 2.0, 5/2007

Outline

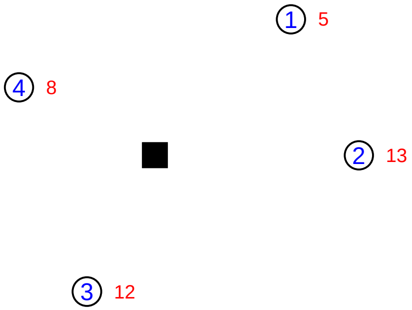
Capacitated Vehicle Routing problem (CVRP): Input Data

Input Data:

- $n = 4$ customers
- 0 depot
- d_i = (0, 5, 13, 12, 8) demands
- Cost Matrix= $\{c_{i,j}\} =$

(i,j)	0	1	2	3	4
0	0	2	3	2	2
1	2	0	2	4	4
2	3	2	0	4.5	5
3	2	4	4.5	0	3
4	2	4	5	3	0

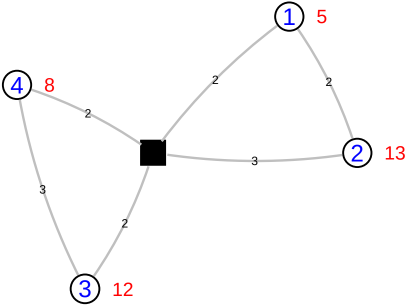
- Q = 20 vehicle capacity



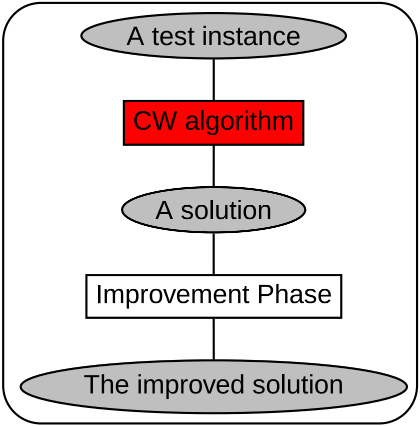
Capacitated Vehicle Routing problem (CVRP): Output Data

Output Data:

- Solution cost: 14
- Routes:
  - **i)** Cost: 7;
    Demand: 18;
    #cust: 2;
    Sequence: 0 1 2 0
  - **ii)** Cost: 7;
    Demand: 20;
    #cust: 2;
    Sequence: 0 3 4 0



The project goal



The Clarke and Wright Algorithm (1964)

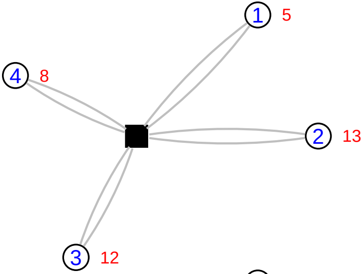
- Clarke and Wright [1964]: Scheduling of vehicles form a central depot to a number of delivery points
- Constructive and greedy heuristic algorithm
- Sequential and Parallel versions (Parallel version performs better, Toth and Vigo [2002])
- Pro :
  - Fast: Complexity: $O(n^2 \log n)$
  - Easy to implement
- Cons : Accuracy
  - Experimental result: +5% respect the best known solutions on benchmark problems
  - Worst case analysis: $CW(I)/OPT(I) <= \lceil \log 2n \rceil + 1$ where:
    - I problem instance
    - $CW(I)$ Clarke and Wright solution value on instance I
    - $OPT(I)$ Optimal solution of instance I

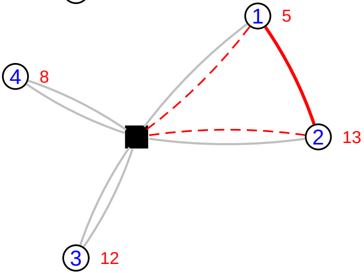
The merge key concept

- Initial solution: each vehicle serves exactly one customer
- The connection (or merge) of two distinct routes can determine a better solution (in terms of routing cost)
- Example:
  We merge routes servicing customers $i = 1$ and $i = 2$. How much do we save?

$$s_{i,j} = c_{i,0} + c_{0,j} - c_{i,j}$$

- If $s_{i,j} > 0$ the merging operation is convenient.





Merge feasibility (1/3)

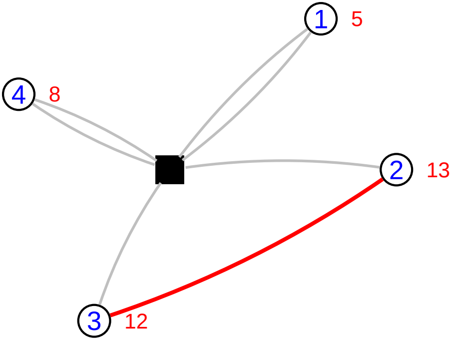
- **Overload of the vehicle**

  The merge operation referred to the customers 2 and 3 in the example is not feasible, in fact:

  - $D_{route} = d(2) + d(3) = 25$
  - $Q = 20$ (vehicle capacity)
  - $D_{route} > Q$

  The route $0 \rightarrow 2 \rightarrow 3 \rightarrow 0$ is not feasible.

  $\Rightarrow$ This merge operation cannot be performed!

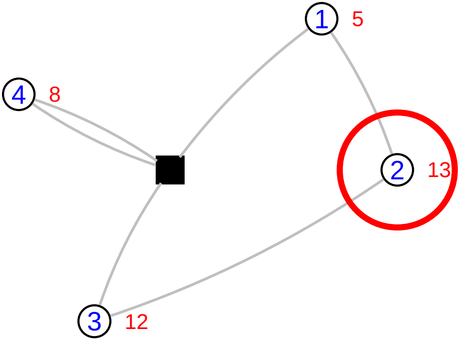


Merge feasibility (2/3)

- **Internal customers**

  A customer which is neither the first nor the last at a route cannot be involved in merge operations.

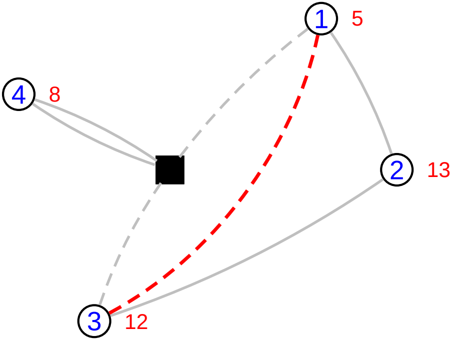
  Example: the customer 2 cannot be involved in any merge operation, because no arc exists connecting 2 to the depot 0.
  $\Rightarrow$ The merge operations suggested by the s_{2j} values cannot be performed!



Merge feasibility (3/3)

- **Customers both in the same route**

If the customers suggested by the saving $s_{i,j}$ are the extremes of the same route (the first or the last) the merge operation cannot be performed (no subtour are allowed)



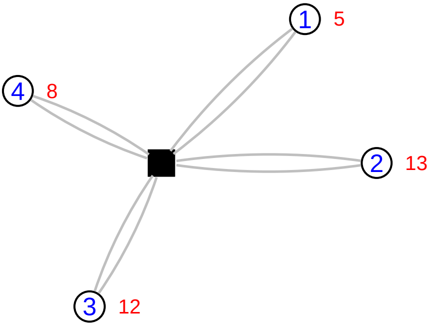
Example: The customer 1 and 3 cannot be involved in any merge operation, because they are in the same route.

$\Rightarrow$ The merge operation suggested by the $s_{1,3}$ value cannot be performed!

Clarke and Wright

The Clarke and Wright algorithm starts as follows:

- The solution is initialized with a route for each customer (**Iteration 0**).
- All the saving values $s_{i,j}, \forall i, j \in 1, \ldots, n \; and \; j > i$ are stored in a half-square matrix M.
- The saving values are ordered in not-increasing fashion in the list L (**the highest saving value the most appealing the merge operation is !**).



Data structure

- We compute for each couple of customers the saving value and we fill the matrix **M** of saving *objects*.
- Each saving *object* is composed by the triplet $(s_{i,j}, i, j)$
- The matrix **M** is sorted respect the $s_{i,j}$ value to create the list **L**, as shown in the example:

Matrix M

	2	3	4
1	3	0	0
2	–	0.5	0
3	–	–	1

$\Rightarrow$

List L

s_{ij}	i	j
3	1	2
1	3	4
0.5	2	3
0	1	3
0	1	4
0	2	4

- The *saving* objects in the list are now sequentially considered: if the associated merge operations are feasible, let's implement them.

Algorithm: iteration 1

List L

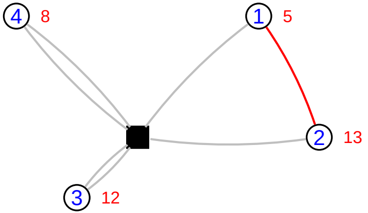
s_{ij}	i	j
3	1	2
1	3	4
0.5	2	3
0	1	3
0	1	4
0	2	4

1. $D_{route} = d(1) + d(2) = 18 < 20 = Q$. OK !
2. Both the customers are extern. OK!
3. The customers 1,2 are not in the same route. OK!

$\Rightarrow$ The merge can be performed: operation feasible.

New solution:

- Solution cost: 11
- Routes:
  - **i)** Cost: 7; Demand: 18; #cust: 2; Sequence: 0 1 2 0
  - **ii)** Cost: 4; Demand: 12; #cust: 1; Sequence: 0 3 0
  - **iii)** Cost: 4; Demand: 8; #cust: 1; Sequence: 0 4 0



Algorithm: iteration 2

List L

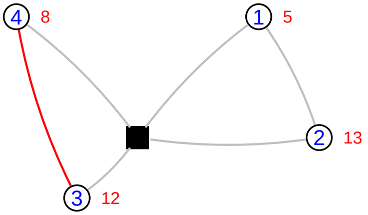
s_{ij}	i	j
3	1	2
1	3	4
0.5	2	3
0	1	3
0	1	4
0	2	4

1. $D_{route} = d(3) + d(4) = 20 = 20 = Q$. OK !
2. Both the customers are extern. OK!
3. The customers 3,4 are not in the same route. OK!

$\Rightarrow$ The merge can be performed: operation feasible.

New solution:

- Solution cost: 10
- Routes:
  - **i)** Cost: 7; Demand: 18; #cust: 2; Sequence: 0 1 2 0
  - **ii)** Cost: 7; Demand: 20; #cust: 2; Sequence: 0 3 4 0



Algorithm: iteration 3

List L

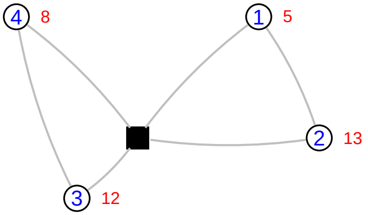
s_{ij}	i	j
3	1	2
1	3	4
0.5	2	3
0	1	3
0	1	4
0	2	4

1. $D_{route} = d(3) + d(2) = 38 > 20 = Q$. **NO!**
2. Both the customers are extern. OK!
3. The customers 3,2 are not in the same route. OK!

$\Rightarrow$ The merge cannot be performed: operation infeasible.

The solution is not updated!!

- Solution cost: 10
- Routes:
  - **i)** Cost: 7; Demand: 18; #cust: 2; Sequence: 0 1 2 0
  - **ii)** Cost: 7; Demand: 20; #cust: 2; Sequence: 0 3 4 0



Algorithm: next iterations

- The next $s_{i,j}$ values in the list **L** are all 0.
- These values correspond to merge operations without a save in the solution routing cost.
- Objective of the algorithm: minimize the number of routes in the solution $\rightarrow$ consider the remaining savings!
- Anyway in the example no more merge operations are feasible, so the algorithm is terminated.

Algorithm: Pseudocode

Algorithm 3.1: CLARKE AND WRIGHT(*InputData*)

$$\begin{cases} \textbf{for } i, j, (j > i) \leftarrow (i = 1, j = 2) \textbf{ to } (i = n-1, j = n) \\ \quad \textbf{do } s_{i,j} \leftarrow c_{0,i} + c_{j,0} - c_{i,j} \qquad !\ \textit{Fill Matrix } M \\ \textit{Sort Matrix } M,\ \textit{filling list } L \\ s_{h,k} \leftarrow \textit{First saving in } L \\ N_{routes} \leftarrow n \\ \textbf{while } ((\textit{List } L\ \textbf{ not } \textit{void}) \textbf{ and } (s_{h,k} > 0)) \\ \quad \textbf{do } \begin{cases} s_{h,k} \leftarrow \textit{First } s_{i,j} \in L \textit{ not yet considered} \\ \textbf{if } (\textit{MergeFeasibility}(h,k) == \textit{YES}) \begin{cases} \textit{Merge}(\textit{Route}_h, \textit{Route}_k) \\ N_{routes} - - \end{cases} \end{cases} \end{cases}$$

Solution Data structure

- Let's introduce a data structure **R** to keep in memory partial solutions during algorithm iterations:
  1. This data structure **R** has to contain routes information.
  2. This data structure has to be useful to implement easily the MergeFeasibility and Merge functions.

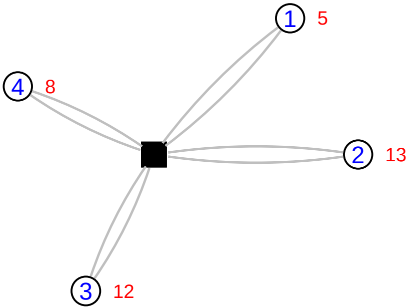


Figure: Iteration 0

Solution data structure **R**

♯*route*	cost	load	♯*cust*	*extreme*$_1$	*extreme*$_2$	Customer sequence
1	4	5	1	1	1	1
2	6	13	1	2	2	2
3	4	12	1	3	3	3
4	4	8	1	4	4	4

Solution Data structure: first iteration

- Is the merge feasible?
  - **i)** $d(2) + d(1) <= 20$? YES
  - **ii)** 2, 1 are both in the extreme list? YES
  - **iii)** 2, 1 are extremes for distinct $\sharp_{route}$? YES

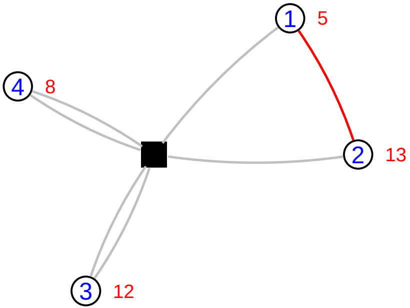


Figure: Iteration 1

Solution data structure **R**

$\sharp_{route}$	cost	load	$\sharp_{cust}$	$extreme_1$	$extreme_2$	Customer sequence
1	4	5	1	1	1	1
2	6	13	1	2	2	2
3	4	12	1	3	3	3
4	4	8	1	4	4	4

Solution Data structure: implement the merge

R update:

- A route has to be deleted.
- In the remaining one, these values have to be updated:
  - **i)** $\sharp_{cust}$
  - **ii)** demand
  - **iii)** sequence (check if a route sequence has to be inverted!!)
  - **iv)** extremes

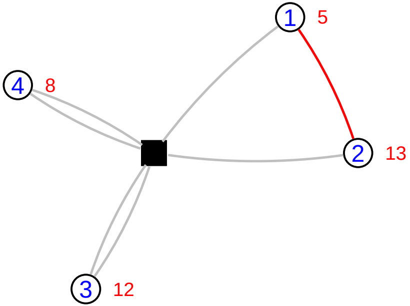


Figure: Iteration 1

Solution data structure **R**

$\sharp_{route}$	cost	load	$\sharp_{cust}$	$extreme_1$	$extreme_2$	Customer sequence
1	7	18	2	1	2	1 2
3	4	12	1	3	3	3
4	4	8	1	4	4	4

Solution Data structure: second iteration

- Is the merge feasible?
  - **i)** $d(4) + d(3) <= 20$? YES
  - **ii)** 4, 3 are both in the extreme list? YES
  - **iii)** 4, 3 are extremes for distinct $\sharp_{route}$? YES

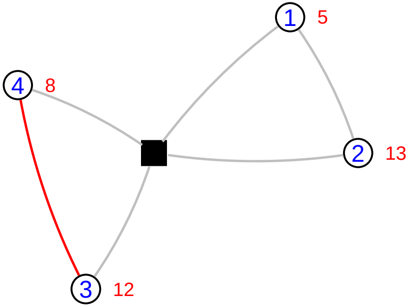


Figure: Iteration 2

Solution data structure **R**

$\sharp_{route}$	cost	load	$\sharp_{cust}$	$extreme_1$	$extreme_2$	Customer sequence
1	7	18	2	1	2	1 2
3	4	12	1	3	3	3
4	4	8	1	4	4	4

Solution Data structure: implement the merge

R update:

- A route has to be deleted.
- In the remaining one, these values have to be updated:
  - i) ♯*cust*
  - ii) demand
  - iii) sequence (check if a route sequence has to be inverted!!)
  - iv) extremes

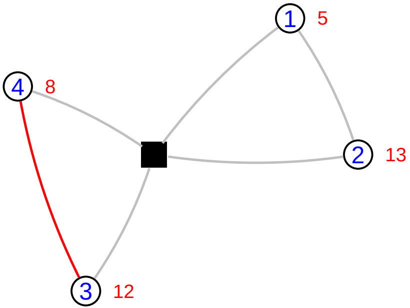


Figure: Iteration 2

Solution data structure **R**

♯*route*	cost	load	♯*cust*	$extreme_1$	$extreme_2$	Customer sequence	
1	7	18	2	1	2	1	2
2	7	20	2	3	4	3	4

Merge operation

How can we join the sequence of customers in a merge operation?

Hp:

- Two routes $A = (1, 2)$, $B = (3, 4)$
- A and B have to be merged in the final route C, according to the saving criterion
- The capacity of the vehicle is ∞.

Four merge situations can occur to obtain the final route C:

1. $s_{3,2} \Rightarrow$ Simple Union deleting route B $C = (1, 2, 3, 4)$
2. $s_{4,1} \Rightarrow$ Simple Union deleting route A $C = (3, 4, 1, 2)$
3. $s_{4,2} \Rightarrow$ Route B has to be inverted $B^* = (4, 3)$, merged to A and then deleted C=(1,2,4,3)
4. $s_{3,1} \Rightarrow$ Route A has to be inverted $A^* = (2, 1)$, route B has to be merged to A^* and then deleted C=(2,1,3,4)

Algorithm Improvement

How can we improve the algorithm performance?

- Accuracy:
  - Multistart approach
  - A parametric saving formula
  - Post-optimization
- Speed:
  - Heap sorting procedure (one or few distinct heaps)
  - Early stop in the saving list L
  - Subset of savings (grid structure)